# IS 6733
# Deep Learning on Cloud Platforms

**Lecture 2c**

**Python Tutorial - Matplotlib**

**Dr. Yuanxiong Guo**

**Department of Information Systems and Cyber Security**

# Python Checklist in Machine Learning

- Essential libraries and tools in data science
  - Jupyter Notebook/Colab
  - NumPy
  - Pandas
  - Matplotlib
  - Scikit-Learn
  - Keras/TensorFlow

# Matplotlib

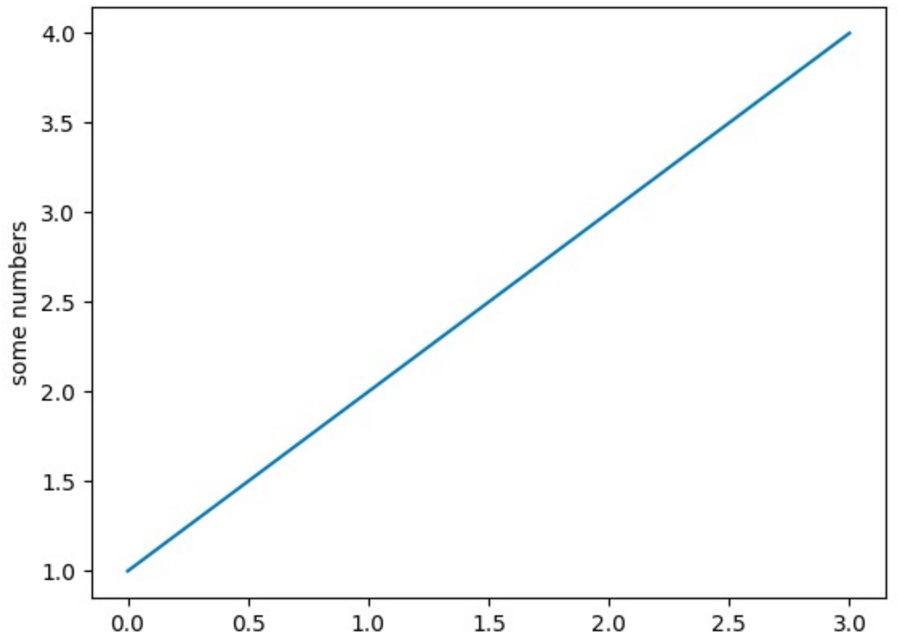
- A Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
- Can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.
- For simple plotting the **pyplot** module provides a MATLAB-like interface – but open source and free
- For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

## Pyplot Module

- Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.
- The most commonly used

# A Simple Example

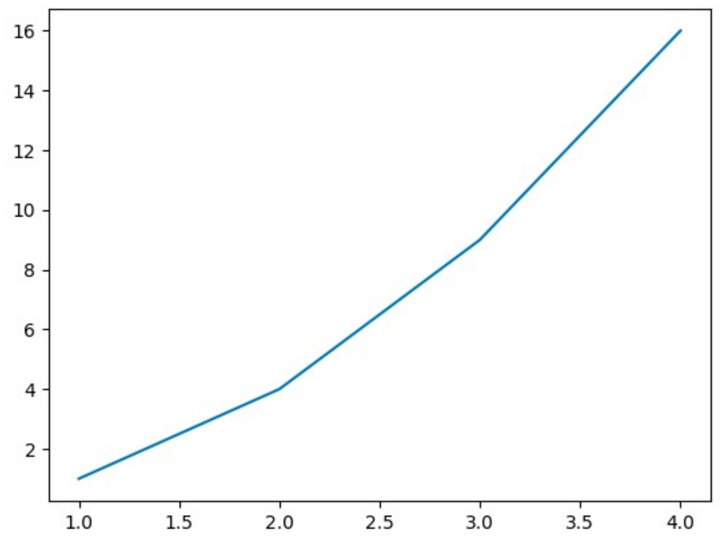
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



- x-axis ranges from 0 – 3, and y-axis ranges from 1 – 4
- If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you.
- Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

## Another Example

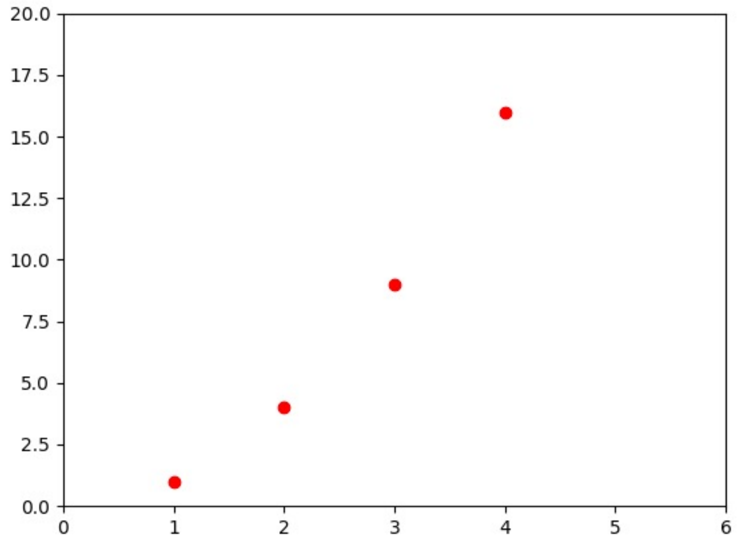
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```



## Formatting the style of your plot

- For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot (similar to MATLAB).
- The default format string is 'b-', which is a solid blue line.

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

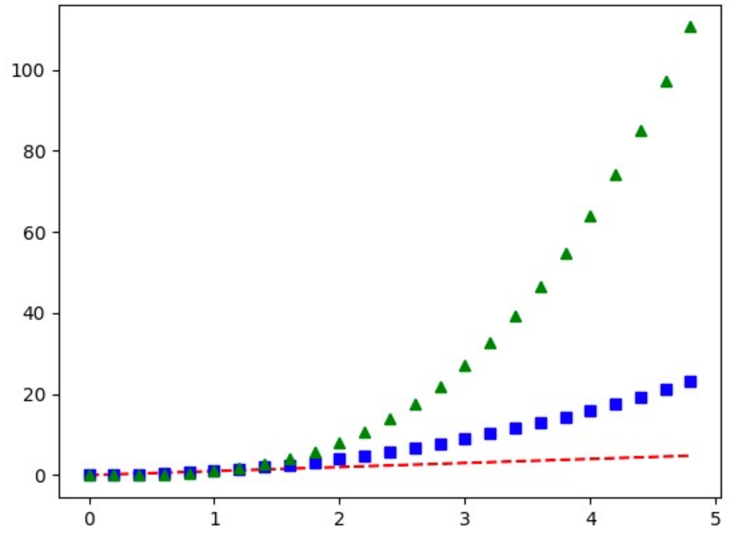


The axis() command in the example above takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

# Used together with Numpy

- Matplotlib is not limited to working with list
- Generally, we will use numpy arrays
- In fact, all sequences are converted to numpy arrays internally in Matplotlib

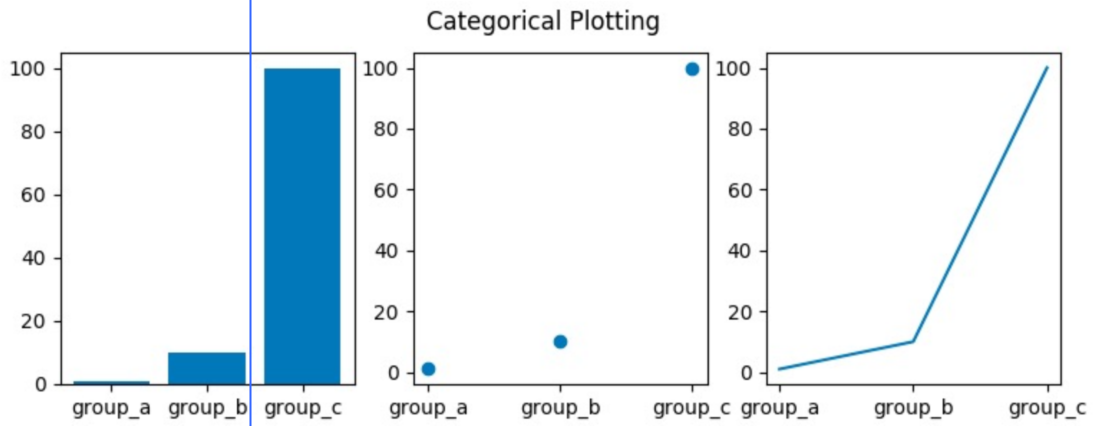
```
import numpy as np

# evenly sampled time at 0.2 intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



# Plotting with categorical variables

- Matplotlib allows you to pass categorical variables directly to many plotting functions.

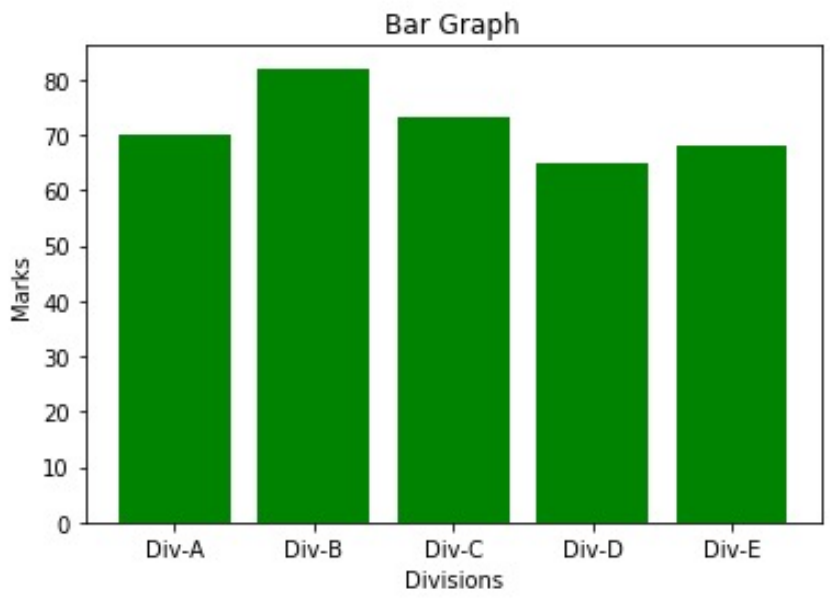
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



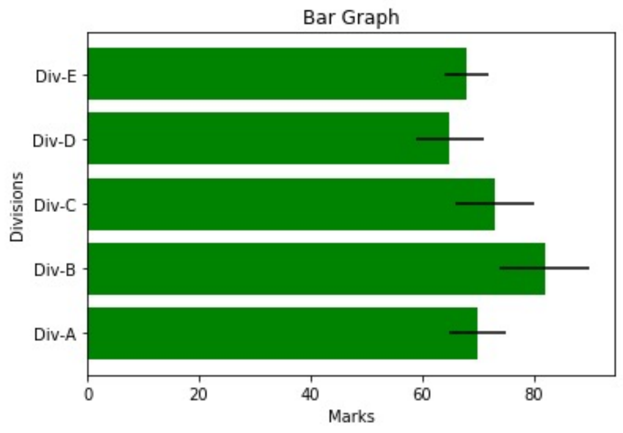
## Controlling Graph Types

- Bar Graphs: one of the most common types of graphs and used to show data associated with the categorical variables.

```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]

plt.bar(divisions, division_average_marks, color='green')
plt.title("Bar Graph")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.show()
```



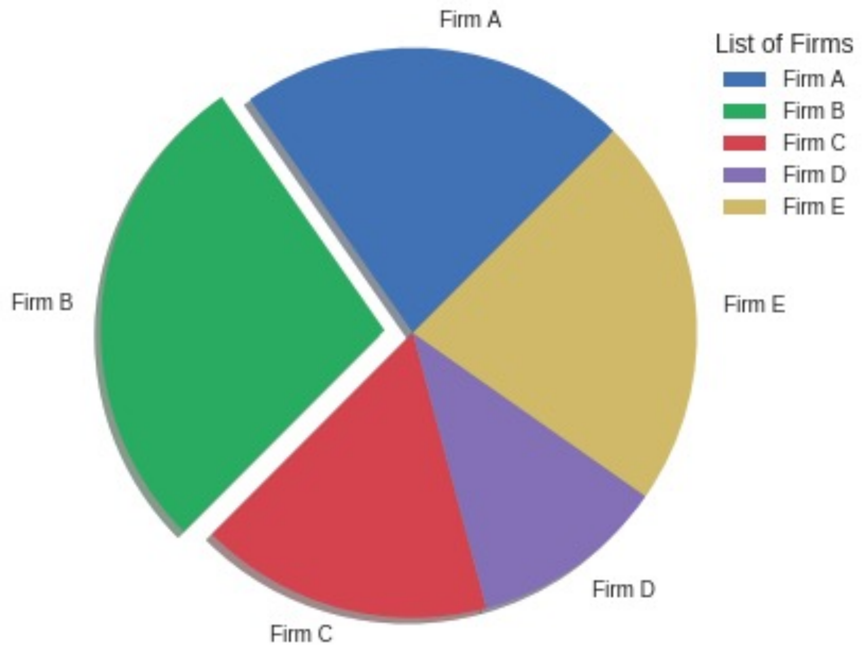
```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]
variance = [5,8,7,6,4]

plt.barh(divisions, division_average_marks, xerr=variance, color='green')
plt.title("Bar Graph")
plt.xlabel("Marks")
plt.ylabel("Divisions")
plt.show()
```



## Controlling Graph Types

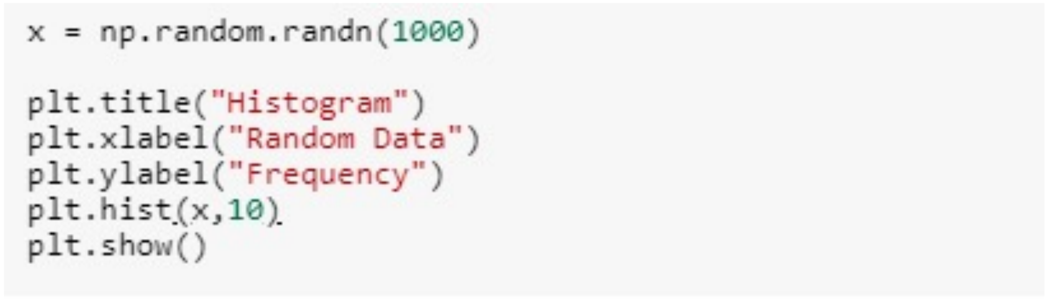
- Pie Charts

```python
firms = ["Firm A", "Firm B", "Firm C", "Firm D","Firm E"]
market_share = [20, 25, 15, 10, 20]
Explode = [0,0.1,0,0,0]
plt.pie(market_share,explode=Explode,labels=firms,shadow=True,startangle=45)
plt.axis('equal')
plt.legend(title="List of Firms")
plt.show()
```

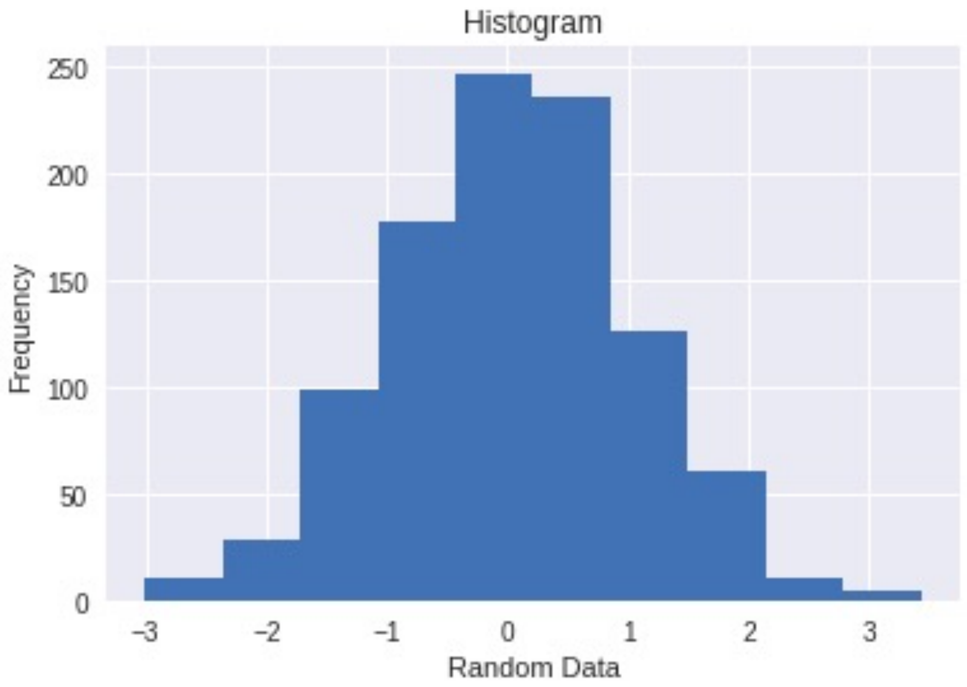


## Controlling Graph Types

- Histogram



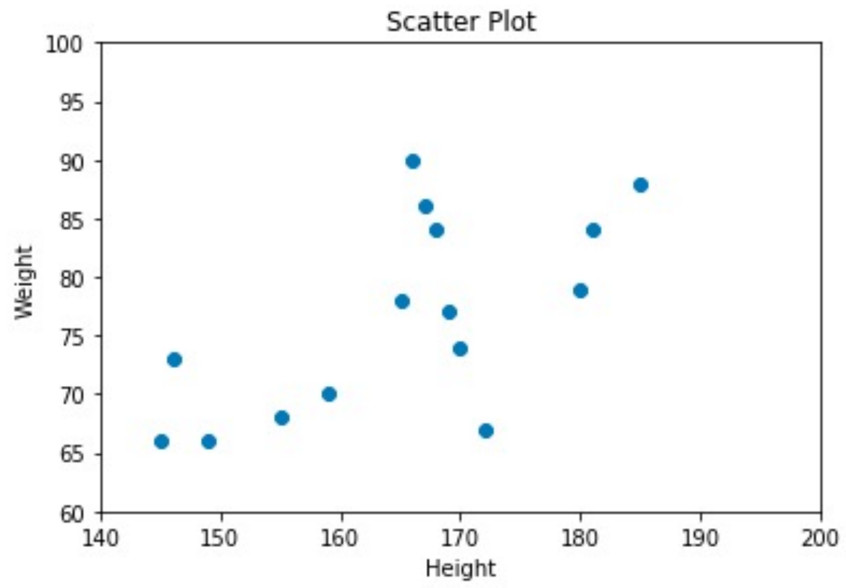
```
x = np.random.randn(1000)

plt.title("Histogram")
plt.xlabel("Random Data")
plt.ylabel("Frequency")
plt.hist(x,10)
plt.show()
```



# Controlling Graph Types

- Scatter Plots

```
height = np.array([167,170,149,165,155,180,166,146,
                   159,185,145,168,172,181,169])
weight = np.array([86,74,66,78,68,79,90,73,
                   70,88,66,84,67,84,77])
plt.xlim(140,200)
plt.ylim(60,100)
plt.scatter(height,weight)
plt.title("Scatter Plot")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.show()
```



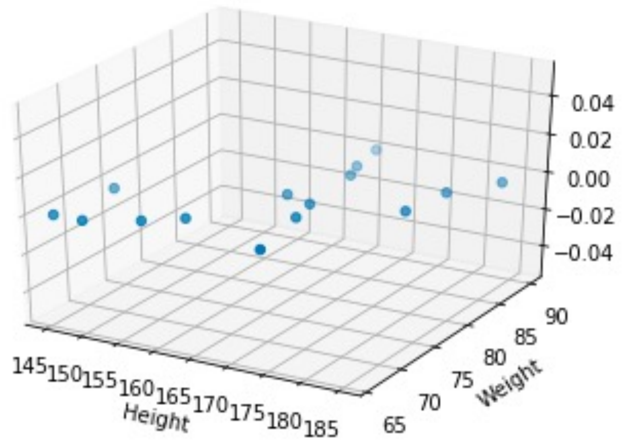
## Controlling Graph Types

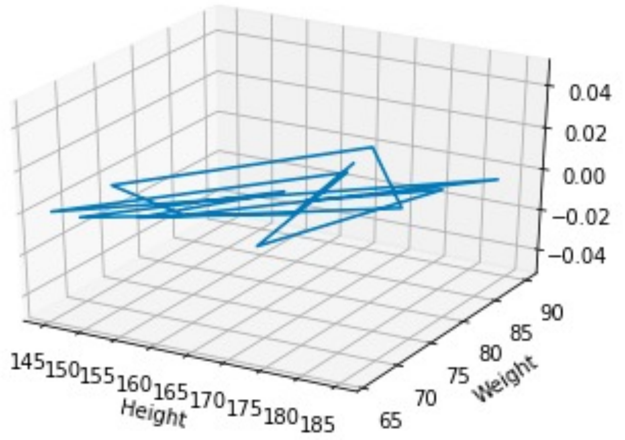
- 3-D Plotting

```
from mpl_toolkits import mplot3d
```

```
ax = plt.axes(projection='3d')
ax.scatter3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```

```
ax = plt.axes(projection='3d')
ax.plot3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```





Plotting Tip:

- For categorical variables utilize Bar/Pie Charts and Boxplots.
- For continuous variables utilize Histograms, Scatterplots, Line graphs, and Boxplots.

## Controlling Line Properties

- Lines have many attributes that you can set: linewidth, dash style, antialiased, etc. There are several ways to set line properties
  - Use keyword args

```
plt.plot(x, y, linewidth=2.0)
```

  - Use the format strings

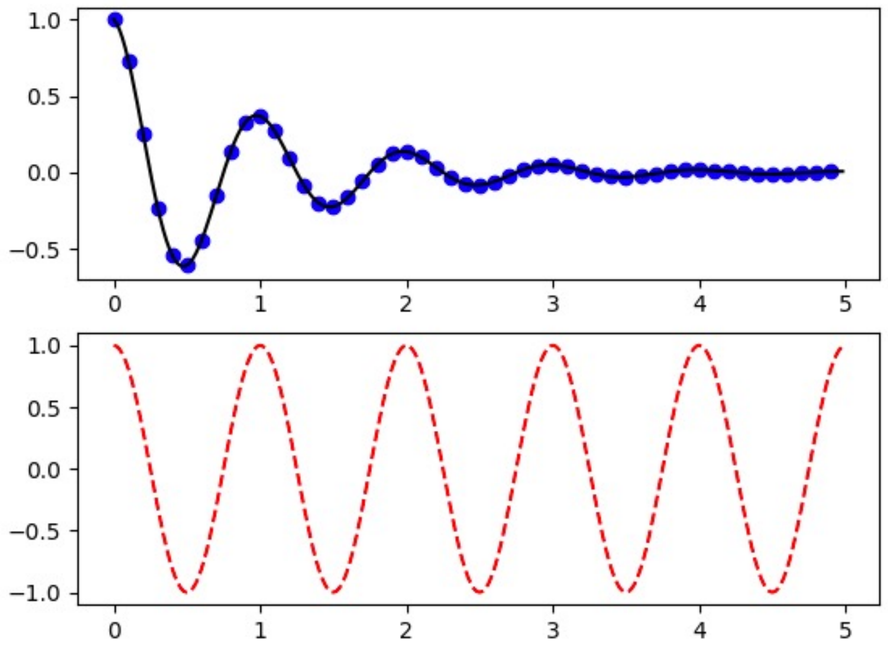
```
plt.plot(x, y, '-')
```

  - Use the setp() command

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

# Working with Multiple Figures and Axes

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



# Working with Multiple Figures and Axes

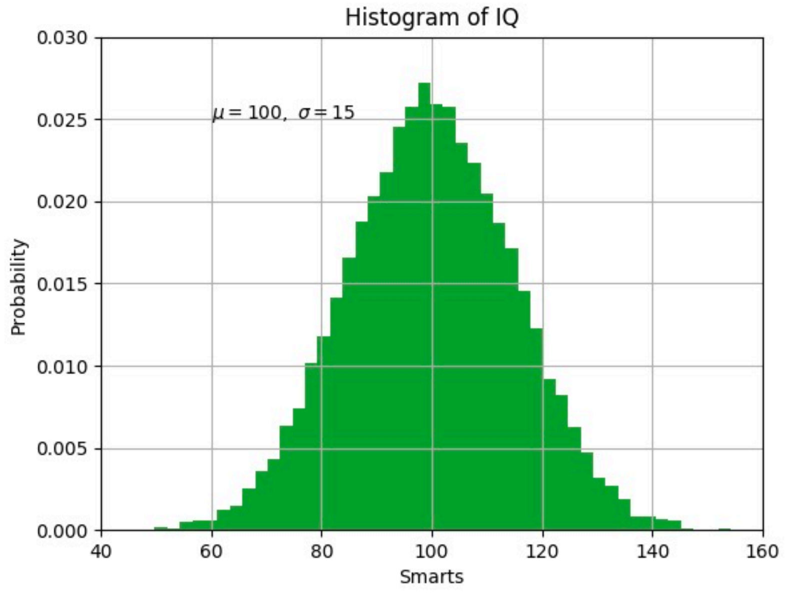
- You can create multiple figures by using multiple figure() calls with an increasing figure number. Of course, each figure can contain as many axes and subplots as you want.

```
import matplotlib.pyplot as plt
plt.figure(1)              # the first figure
plt.subplot(211)           # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)           # the second subplot in the first figure
plt.plot([4, 5, 6])

plt.figure(2)                # a second figure
plt.plot([4, 5, 6])          # creates a subplot(111) by default

plt.figure(1)                # figure 1 current; subplot(212) still current
plt.subplot(211)             # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
```

## Working with text

- The text() command can be used to add text in an arbitrary location, and the xlabel(), ylabel() and title() are used to add text in the indicated locations

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1,
facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



```
t = plt.xlabel('my data', fontsize=14, color='red')
```

# Logarithmic and other nonlinear axes

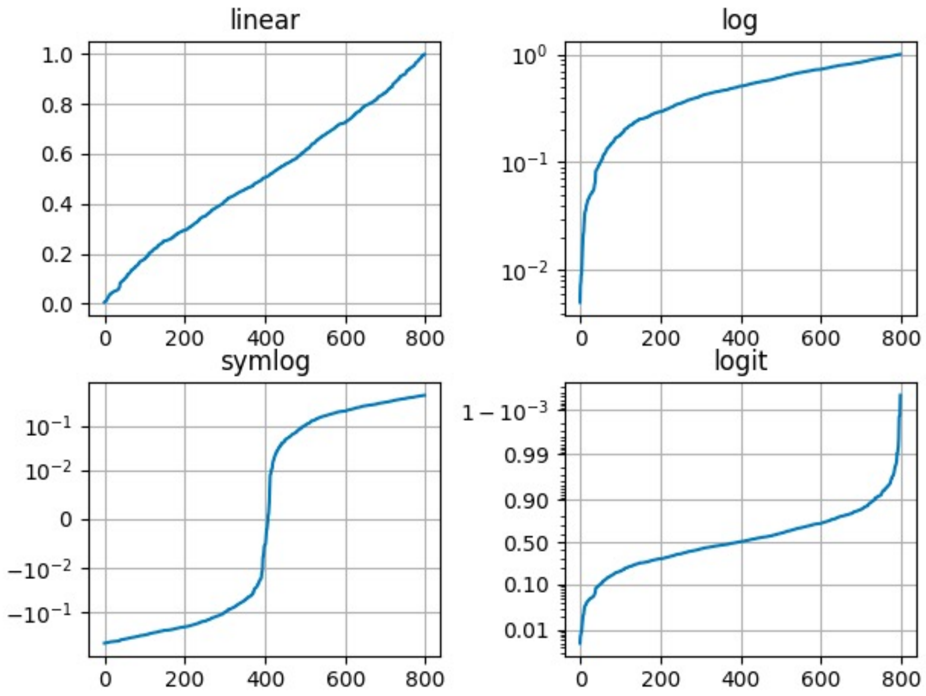
- matplotlib.pyplot supports not only linear axis scales, but also logarithmic and logit scales. This is commonly used if data spans many orders of magnitude.

*# linear*
plt.yscale('linear')

*# log*
plt.yscale('log')

*# symmetric log*
plt.yscale('symlog', linthreshy=0.01)

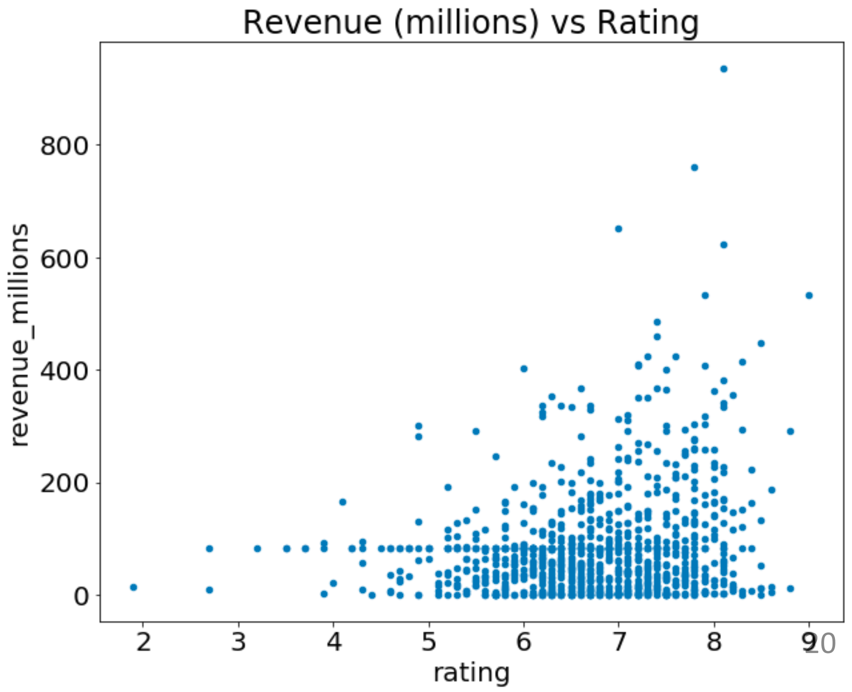
*# logit*
plt.yscale('logit')



# Integrate Matplotlib with Pandas

- A great thing about Pandas is that it integrates with Matplotlib, so you get the ability to plot directly off DataFrames and Series.
- Using the DataFrame example before, let's plot the relationship between ratings and revenue. All we need to do is call .plot() on movies_df with some info about how to construct the plot:

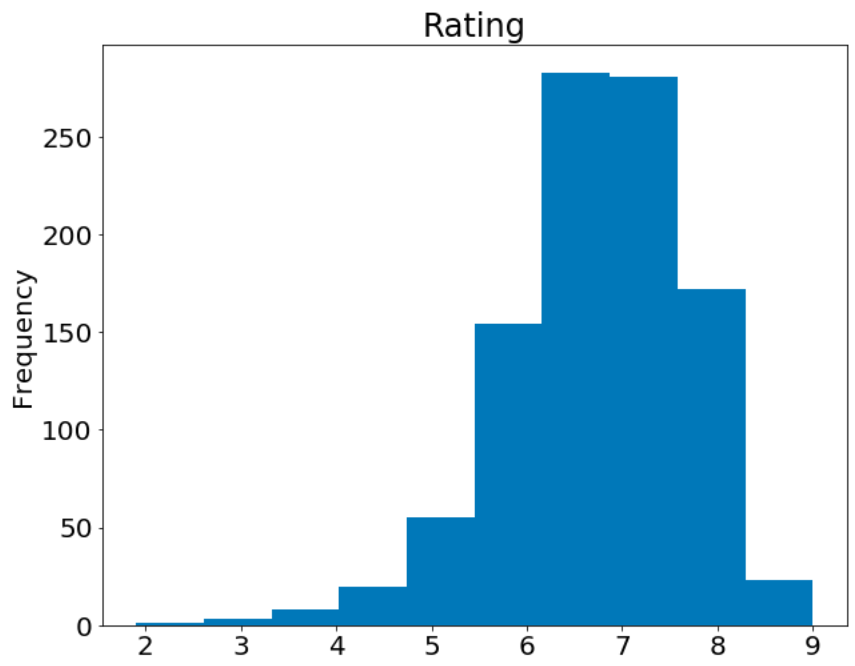
```
movies_df.plot(kind='scatter', x='rating',
y='revenue_millions', title='Revenue
(millions) vs Rating');
```



# Integrate Matplotlib with Pandas

- If we want to plot a simple Histogram based on a single column, we can call plot on a column:

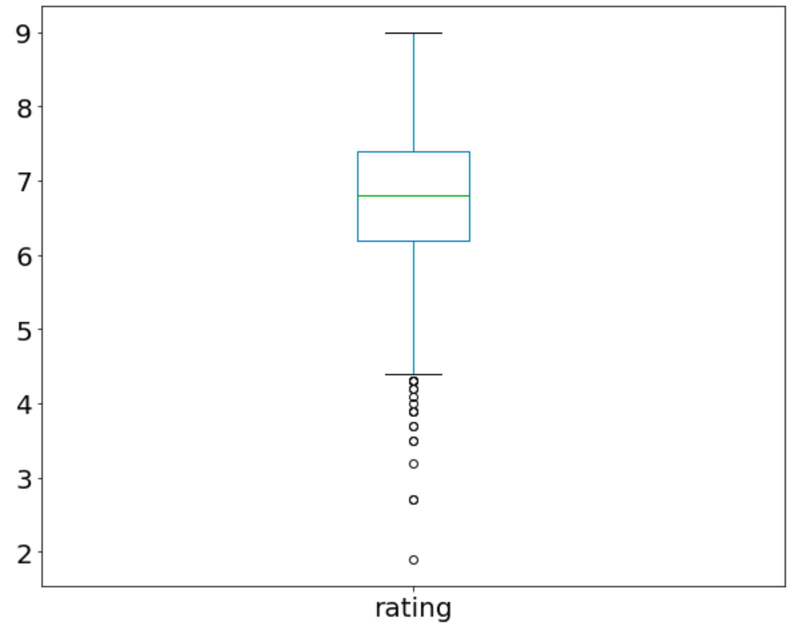
```
movies_df['rating'].plot(kind='hist',
title='Rating');
```

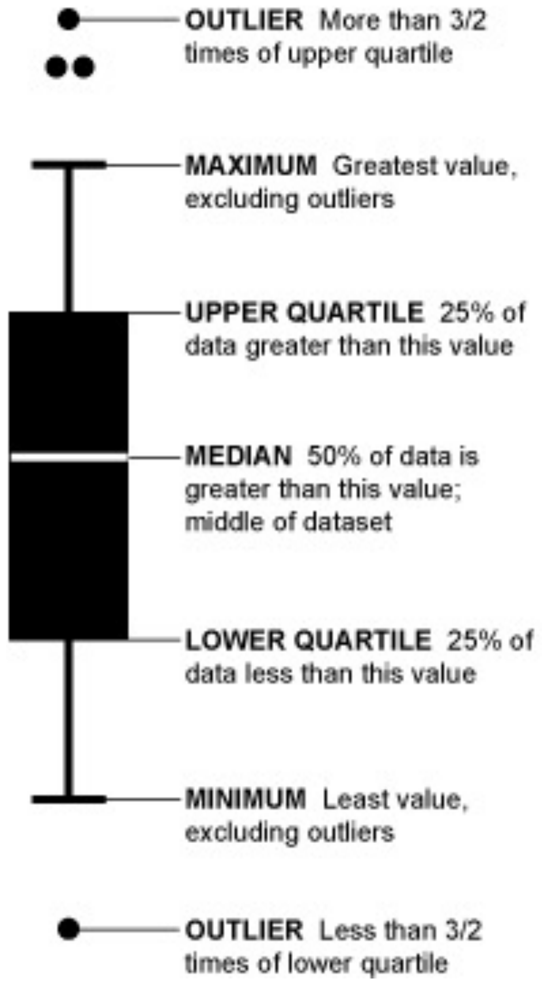


# Integrate Matplotlib with Pandas

- We can also visualize this data using a Boxplot

```
movies_df['rating'].plot(kind="box");
```





OUTLIER More than 3/2 times of upper quartile
MAXIMUM Greatest value, excluding outliers
UPPER QUARTILE 25% of data greater than this value
MEDIAN 50% of data is greater than this value; middle of dataset
LOWER QUARTILE 25% of data less than this value
MINIMUM Least value, excluding outliers
OUTLIER Less than 3/2 times of lower quartile

## Further References

- Seaborn provides an API on top of Matplotlib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas DataFrames.

- Chapter 4 of [T1]